LBNL-47653

ASSESSING MULTIZONE AIRFLOW SOFTWARE

David M. Lorenzetti
dmlorenzetti@lbl.gov

Environmental Energy Technologies Division
Indoor Environment Department
Lawrence Berkeley National Laboratory
Berkeley CA, USA

December 2001

This work was supported by the Office of Nonproliferation Research and Engineering, Chemical and Biological National Security Program, of the National Nuclear Security Administration under U.S. Department of Energy Contract No. DE-AC03-76SF00098.

Assessing Multizone Airflow Software

David M. Lorenzetti
Lawrence Berkeley National Laboratory, Berkeley CA 94720
Report LBNL-47653
December 2001

Abstract

Multizone models form the basis of most computer simulations of airflow and pollutant transport in buildings. In order to promote computational efficiency, some multizone simulation programs, such as COMIS and CONTAM, restrict the form that their flow models may take. While these tools allow scientists and engineers to explore a wide range of building airflow problems, increasingly their use has led to new questions not answerable by the current generation of programs.

This paper, directed at software developers working on the next generation of building airflow models, identifies structural aspects of COMIS and related programs that prevent them from easily incorporating desirable new airflow models. The paper also suggests criteria for evaluating alternate simulation environments for future modeling efforts.

Keywords: airflow, COMIS, multizone

1. Introduction

Multizone models form the basis of most computer simulations of airflow and pollutant transport in buildings. The state of the art in whole-building flow analysis relies on these lumped-parameter, or macroscopic, models, because representing an entire building microscopically (e.g., using computational fluid dynamics) would require unrealistic amounts of input data, computer memory, and processing power. This situation will likely persist for some time.

This paper assesses current multizone simulation programs, in light of modeling applications of interest to the Airflow and Pollutant Transport group at Lawrence Berkeley National Laboratory. To ground the discussion, it mainly focuses on a particular code, COMIS. The fact that this code was developed at LBNL gives us greater insight into its architecture, and greater latitude to criticize it. However, much of the discussion applies generally to other multizone airflow and pollutant transport programs.

Organization. Section 2 provides an overview of the present critique of multizone simulation software. It discusses the rationale for reviewing the state of the art in multizone simulation, and summarizes the arguments contained in the rest of the paper.

Sections 3 and 4 provide background material for the critique itself. Section 3 discusses general software characteristics of interest to multizone modelers, defining some terms used in the later comments on specific simulation programs. Section 4 briefly describes COMIS, and defines a sample flow component—a duct—for later reference.

Section 5 critiques multizone software tools, both in terms of existing flow models, and in terms of their ability to incorporate new models. Section 6 considers the prospects for modifying COMIS to answer these objections, and proposes key features to look for in a new environment.

Unless otherwise noted, "COMIS" refers to COMIS v. 3.0.2, the last freely-available release of the source code to the program [7].

2. Overview

The design of the current generation of multizone programs results from a systematic drive toward computational efficiency. This effort was founded in the desire to bring whole-building airflow simulation to ordinary desktop computers, with their limited memory and computing power. Therefore the data structures and solution algorithms of these programs were optimized for a fairly specific task—briefly, that of solving a steady-state flow network in which the flow through each path increases monotonically with the pressure drop across it.

Partly because of this specialization, COMIS and programs like it have allowed researchers and engineers to explore a wide range of building airflow problems. However, these programs now can

be seen as victims of their own success. The insights gained through the use of these tools have naturally led to questions that probe beyond the problem space they were designed to explore.

This situation, along with the availability of faster, more capable computers, makes a less specialized, and more flexible, software tool increasingly attractive.

An important implementation question involves whether to design a more general multizone simulation tool "from scratch," or to structure it around an existing program. This paper argues that a number of factors make COMIS unsuitable for the long-term needs of users interested in modeling complex phenomena such as pollutant dispersion times, feedback control systems, and natural ventilation.

These factors fall into two broad categories. First, COMIS lacks specific models necessary for certain kinds of simulations. Second, and more importantly, structural limitations in the program limit its ability to accomodate new models.

2.1. Missing models. As researchers attempt to simulate buildings in which effects such as natural convection, novel mechanical systems, automatic controls, and transport delays in flow paths play an important part, they increasingly require a number of models not currently supported by COMIS.

Such models include: (1) buoyancy-induced two-way flows through horizontal apertures, including stairwells and elevator shafts; (2) "lossy" duct junctions, which provide the correct pressure drops at all flows; (3) feedback control elements, such as sensors, controllers, and actuators; (4) dynamic flow components, including both the time needed to carry pollutants through flow elements, and transient airflow effects; and (5) detailed models of the air movement in large spaces, for example using coarse-grid computational fluid dynamics (CFD).

Minor modifications would allow COMIS to support some of these models. However, it cannot support others. For example, it could accomodate a simple model for bidirectional flows in horizontal apertures, subject to the provisions described below. However, the standard model for duct junctions violates a fundamental design restriction of the program.

Other flow models have not yet been characterized well enough to permit a definitive statement about incorporating them into COMIS. However, due to structural limitations in the code, nothing less than a complete overhaul would make it accomodate all of these models. More to the point, adding those models that COMIS can support without a major rewrite would only increase the pressure from modelers to add one of the proscribed models. Hence, these modeling limitations must be addressed eventually—either inside or outside the framework provided by the current program.

2.2. Structural limitations. Any major overhaul of COMIS should address the following structural issues: (1) incomplete coupling of the energy balance to the flow element relations; (2) non-repeatable flow calculations; (3) restrictions on the mathematical form of flow element and zone models; (4) tight coupling between the parts of the code that define flow systems, and the parts that solve flow systems; and (5) a code base that induces slow turn-around times for creating, testing, and modifying new models.

These structural issues mainly result from design decisions taken to optimize the code. In such cases, the specialized data structures and solution algorithms employed also act to restrict the form that models can take. In addition, unintended restrictions arise due to details of the program implementation.

Arguably, the effort needed to overhaul the current code would be better spent writing a replacement tool, or reproducing the current capabilities in an existing general simulation environment. This conclusion follows both from the extent to which structural limitations pervade the existing code, and from the difficulty of recasting that code into a structured programming style.

3. Modular Simulation

The multizone airflow literature usually describes COMIS, and simulation environments like it, as "modular." Before examining specific problems with these programs, it helps to distinguish between modularity as perceived by casual users, by programmers, and by researchers. This section makes some general points about modular environments, in preparation for the later discussion of COMIS.

3.1. Modularity. In macroscopic modeling, a *modular* simulation environment allows the user to construct complex systems by assembling relatively simple component models. The system's behavior arises from the composite interaction of its components. By contrast, a *monolithic* tool defines the system response integrally.

Many simulation programs claim some degree of modularity. Examples from the building airflow and energy fields include COMIS, CONTAM, HVACSIM+, and EnergyPlus. For electronics applications, the modular program SPICE represents circuits as collections of resistors, capacitors, transistors, and so on.

At their best, modular environments aid simulation by decomposing a system into manageable, verifiable, and interoperable submodels. This can provide greater flexibility than a monolithic tool, by allowing the user to propose new combinations of existing component models. It also eases the task of adding new components to the simulation environment.

Modularity does not, however, guarantee flexibility in the models themselves. Like all simulation environments, modular programs support only certain types of components and component connections. Usually the restrictions follow from implementation issues—such as limitations of the programming language, or, more often, the decision to improve efficiency at the expense of flexibility. For example, COMIS requires every flow to depend on the pressure difference between two zones [14]. This guarantees certain "nice" numerical properties, but prevents the program from accurately representing some real-world flow components, such as duct junctions.

3.2. Weak versus strong modularity. In practice, modular environments can exhibit monolithic behavior for certain aspects of the system. This occurs whenever the program makes implicit assumptions about the models—as part of the solution algorithm, for example. Although in theory the system behavior results solely from interactions between the modular components, monolithic code adds interactions between the component models and the simulation environment. Unfortunately, these monolithic assumptions often remain hidden from the user.

For example, COMIS is modular in that the user assembles an airflow network by connecting zones via flow elements. However, the program distributes the calculations associated with the zones throughout the code that solves the airflow system. Thus the air density calculations lie outside the reach of any model explicitly provided to the user, and COMIS enforces a monolithic zone model. Furthermore, the program uses a common subroutine to calculate the hydrostatic effects associated with every flow path. To this extent, it enforces monolithic behavior for part of every flow model.

To formalize this distinction, define a *weakly modular* simulation environment as one that simply treats its components as building blocks for creating system models. A *strongly modular* environment, on the other hand, represents every component as a collection of independent algorithms. The algorithms are independent in two senses. First, taken together, they completely specify the component's behavior. Second, removing a component's algorithms from a strongly modular simulation environment does not affect the behavior of any other component.

Note that the distinction between strong and weak modularity concerns only where in code the models reside, without reference to the severity of the restrictions placed on the models.

In the coding of a computational tool, weak modularity often manifests itself in: (1) the distribution of a component model's definition across a number of programming modules; (2) extensive generic preprocessing or postprocessing of the data associated with the component models; (3) case-based logic that invokes special processing depending on the status of one or more components; and (4) the merging of the solution algorithm with the component routines.

These practices obscure the actual modeling assumptions made in a weakly modular program. They also complicate the process of updating or adding components, because they expose supposedly stable or mature component models to the possibility of unintended changes. Consequently, they require the programmer to have deep familiarity with the internal workings of the environment. Finally, weak modularity complicates the inclusion of new solution algorithms, to the extent that it requires new solution methods to implement the modeling assumptions made in the existing algorithms.

3.3. Modularity and user expectations. Casual users, programmers, and researchers all place different demands on a simulation tool.

A casual user, interested in flexible system-level modeling rather than in the behavior of individual components, demands only weak modularity (and some assurance that the models perform as advertised).

From a programmer's point of view, however, weak modularity does not imply flexibility, since the implementation may achieve its component orientation in roundabout ways—for example, using extensive patches of monolithic code to modify the behavior of its internal algorithms on a component-by-component basis.

The defects of weak modularity affect the research user as well as the programmer of a simulation tool. Weak modularity prevents researchers from easily identifying the assumptions behind a model, from specifying and testing changes to an existing model, and from adding new models without fear of altering the behavior of existing ones.

Strong modularity counters these objections. However, a strongly modular simulation tool does not necessarily provide the researcher with complete flexibility in adding new components. Even a strongly modular environment can restrict the form of components and their interconnections. For instance, COMIS could be recoded in a strongly-modular fashion, yet still only support flow models whose flows depended on the pressure difference between two zones. Thus, it would remain unable to simulate duct junctions accurately.

Of course, restricting the types of models supported by a simulation environment does not have all negative effects. As mentioned above, it can promote computational efficiency. Furthermore, such restrictions can force the user to clarify his or her thinking about the models and the modeling process. For example, system dynamics programs do not normally allow algebraic loops. Besides reducing the computational burden of running the model, this restriction forces the user to focus on the important dynamic effects in the system of interest [8 §7].

4. Sample Flow Element

Multizone models idealize a building as a collection of zones (e.g. rooms), connected by discrete flow paths. COMIS solves a steady-state airflow system by balancing flows so that the total inflows and outflows of air for each zone sum to zero [14]. The flows are assumed to have no effect on the zone temperatures. After finding the flows, the program can solve a dynamic contaminant transport problem.

Flow elements dominate multizone models, in two ways. First, multizone programs predict the flows in each path, but ignore details of flow within rooms. Thus, a user adds richness and complexity to a building model mainly by selecting from among a variety of flow element models. Second, the majority of the modeling restrictions in multizone simulation programs apply to the flow models [14].

Because many of the specific problems in COMIS relate to its handling of flow elements, this section defines a specific flow model, a duct, to provide a point of reference.

4.1. Mechanical energy balance. Consider a COMIS flow element that connects two zones, 1 and 2. The component model relates the mass flow leaving zone 1, $f_{1\text{-}2}$, to the pressures p_1 and p_2 at the component connections (note that the simple notation used here does not extend well to multiple flow elements).

Neglecting kinetic energy, a mechanical energy balance on the flow element yields [10 §5.4]

$$p_1 + \rho g z_1 = p_2 + \rho g z_2 + \Delta p_L \ , \tag{4.1}$$

where: (1) ρ gives the density of air in the flow element; (2) z_1 and z_2 give the absolute heights of the element's connections to zones 1 and 2, measured from a global reference level; and (3) Δp_L expresses the mechanical energy dissipated by viscous and dynamic effects. With flow from zone 2 to zone 1, $f_{1\text{-}2} < 0$, Δp_L becomes negative.

4.2. Duct model. Every flow component defines its own relation between the flow and the dissipated mechanical energy. For instance, a steady-state Darcy-Weisbach duct model [10 §7.2] uses

$$\Delta p_L = \text{sign}\{V_{1\text{-}2}\} \lambda \frac{L}{D} \left(\frac{\rho V_{1\text{-}2}^2}{2} \right) \tag{4.2}$$

for a duct of length L and diameter D. Since the mass flow varies directly with the average velocity, let $V_{1\text{-}2}$ replace $f_{1\text{-}2}$ as the variable of interest. The friction factor, λ, comes from the Hagen-Poiseuille and Colebrook relations:

$$\lambda = \frac{64}{\mathrm{Re}} \qquad \text{if} \qquad \mathrm{Re} < 2300 \tag{4.3a}$$

$$\frac{1}{\sqrt{\lambda}} = -2.0 \log\left(\frac{\epsilon}{3.7D} + \frac{2.51}{\mathrm{Re}\sqrt{\lambda}}\right) \qquad \text{if} \qquad \mathrm{Re} \geq 2300 \tag{4.3b}$$

where the Reynolds number $\mathrm{Re} = \rho|V_{1\text{-}2}|D/\mu$. Note that the Colebrook relation defines the friction factor implicitly, that is, it cannot be solved explicitly for λ. For simplicity, ignore the discontinuity in λ at $\mathrm{Re} = 2300$.

Finally, a path density relation

$$\rho = \rho\{V_{1\text{-}2},\, \rho_1,\, \rho_2\} \tag{4.4}$$

reflects the fact that, as $V_{1\text{-}2}$ changes from large positive to large negative values, the steady-state density of air in the duct changes from zone density ρ_1 to ρ_2. In practice, Equation 4.4 cannot be inverted to give velocity in terms of the path density, because: (1) a range of velocities can set $\rho = \rho_1$ (or ρ_2); and (2) identical zone temperatures give the same ρ, regardless of the velocity.

Equations 4.1 through 4.4 couple more tightly than suggested by casual inspection. The path density ρ appears in every equation, and the flow velocity appears in three of them.

A typical simulation treats z_1, z_2, L, D, ϵ, μ, ρ_1, and ρ_2 as parameters. This leaves four equations in six unknowns: p_1, p_2, Δp_L, ρ, $V_{1\text{-}2}$, and λ. A well-posed simulation problem must specify two of the unknowns. COMIS always resolves the choice by having the solution algorithm choose the pressures. The duct component model calculates the resulting flow.

5. Critique

A number of concerns raise doubts about using COMIS as the starting point for a more flexible multizone simulation program. Some follow from its weak modularity, with all the difficulties that implies for understanding, updating, and adding new components. Others follow from design decisions made to improve efficiency in the original program. Still others relate to purely practical implementation issues.

Concerns related to weak modularity include:

- It decouples the mechanical energy balance from the flow element equations.
- It does not provide interchangeable zone models.
- It does not make an effective model development environment.

Concerns related to design decisions made to improve efficiency include:

- Flows must take the form $f_{1\text{-}2} = f_{1\text{-}2}\{p_1 - p_2\}$.
- Airflow models cannot include dynamic effects.
- Flow components give nonrepeatable results due to "memory."

Practical implementation concerns include:

- The code uses static memory allocation, and is not object-oriented.
- The COMIS license agreement complicates questions of ownership.

The following subsections treat these points in greater detail.

5.1. Decoupled mechanical energy balance. The program does not solve the mechanical energy balance together with the flow element equations. In the case of the duct, this means it does not simultaneously satisfy all of Equations 4.1 through 4.4. Hence, it can calculate incorrect flows.

The equations defining the duct couple to the mechanical energy balance through two variables: the path density, ρ, and the dissipated energy, Δp_L. COMIS ignores this coupling, by: (1) setting the path density in Equation 4.1 to some average of the zone densities ρ_1 and ρ_2; and then (2) using the mechanical energy balance to find Δp_L. It does this without regard for the flow velocity—in the case of the duct, ignoring Equation 4.4.

Inspecting the mechanical energy balance shows that this procedure finds an incorrect hydrostatic effect, $\rho g(z_1 - z_2)$, for any vertical path between two zones of different density. That is, unless $\rho_1 = \rho_2$, or unless $z_1 = z_2$, COMIS can find an incorrect value for Δp_L. Consequently the duct routine, which treats the resulting dissipation as a fixed input, can find an incorrect flow. In fact, since Δp_L may even have the wrong sign, the calculated flow may go in the wrong direction.

The calculation of Δp_L takes place in a piece of monolithic code, meaning no flow element couples to the mechanical energy balance correctly. This simplifies the flow component routines—they no longer have to solve all the equations simultaneously—but can yield incorrect flows.

Fortunately, the most significant errors are likely to occur when the driving pressure drop across an element, $|p_1 - p_2|$, is small compared to the hydrostatic pressure through the vertical path, $\rho g|z_1 - z_2|$. Thus, the calculation errors will appear mainly in situations of low flow. However, for vertical paths of large cross-sectional area, such as stairways, elevator shafts, and passive ventilation stacks, these flows (and hence the absolute error) can be quite large. Appendix A gives a numerical example that suggests errors in the neighborhood of 30%.

Bidirectional vertical flows. As a particular consequence of decoupling the mechanical energy balance from the flow equations, COMIS cannot implement any meaningful model for bidirectional flow between floors of a building. Bidirectional flow can occur when the buoyancy-induced pressure difference across a horizontal partition is significant compared to any mechanically-imposed pressures [12]. When the imposed pressure difference does not dominate the calculation of Δp_L, the flow model must solve for the dissipated energy and the mass flow simultaneously.

Fixing the problem. Correcting the program's treatment of the energy balance could follow one of two broad approaches: (1) restructure the code to couple all the flow models correctly; or (2) patch the monolithic part of the code so that it correctly couples any critical flow elements, such as stairways and horizontal apertures.

These two alternatives illustrate one reason a weakly modular program tends to accumulate "special case" programming over time. The first approach would strengthen the code's modularity, and make the entire program more technically correct. However, it would also demand a large programming effort, complicate every flow routine, and in many cases produce no change in the calculated flows. The second approach would require less programming. However, it would make the monolithic routine more difficult to interpret and modify, and would place a greater burden on system modelers to choose the appropriate flow models when representing vertical flow paths.

5.2. No modular zone models. COMIS provides interchangeable models only in the flow components. Zones have exactly one, built-in, idealization—as a well-mixed space. Furthermore, the airflow system represents that space using exactly one state variable (the pressure at a reference height), from which it calculates all other zone data by applying the hydrostatic principle and any user-defined temperature distribution. For every contaminant, the pollutant transport model adds another state variable to each zone.

This monolithic zone model appears: (1) in the airflow network solver, which seeks the reference pressures that produce mass balance in each zone; (2) throughout the code that calculates pressures and densities at the points where flow elements connect to the zones, including the decoupled energy balance; and (3) in the code that finds the pollutant transport between zones. This implementation affects both the ease of adding new zone models, and the maintainability of the code.

New zone models. Providing only one zone model simplifies the program's structure [14], but restricts the types of systems the program can simulate. Situations that would require more general zone models include the need to resolve flows within a room, and the desire to model pollutants with significantly different density than air. Implementing these more general models without changing the internal structure of COMIS would require breaking a room up into a collection of well-mixed subzones, each completely characterized by the given state variables.

The first case—computing flows within a room—arises for example in an atrium or auditorium, where details of pollutant transport in the room determine occupant exposures, and affect the placement and performance of sensors. Two competing approaches, zonal models and computational fluid dynamics, can both represent a room as an interconnected set of well-mixed subzones. Coarse-grid CFD predicts flow velocities better than zonal models of the same arithmetic complexity [16].

However, coarse-grid CFD cannot be formulated directly in COMIS, because the airflow system would require more than a pressure state variable to characterize each subzone*.

The second case demanding a detailed zone model—pollutant stratification—involves for example the transport of smoke, or of dense pollutants. In this case, finding air velocities in the zone matters less than establishing the vertical distribution of pollutant. Modeling these effects would require changing the code that calculates pollutant transport between zones, and possibly the code that accounts for hydrostatic effects in the zones.

For detailed zone models that break up a room into well-mixed subzones, note that the flows between subzones also have to conform to the restrictions imposed by COMIS, in order to implement the desired behavior without changing the program. A later section takes up the modeling requirements COMIS imposes on flow paths, and their implications for zonal models.

Maintainability. Because the monolithic COMIS zone implemention appears throughout the code that solves the airflow and pollutant transport systems, it also makes it hard to modify the solution techniques. That is, under the present design, any new solution algorithm—introduced, say, to improve the speed, robustness, or memory requirements of the program—would have to reproduce the current zone representation. Besides increasing the programming complexity, this could lead to a situation in which the simulated system behavior depends on the particular solver a user chooses to employ.

5.3. Not a model development environment. Due to its weak modularity, COMIS complicates the development of new flow models, and lacks transparency as a research tool. As pointed out above, weak modularity obscures the assumptions behind the component models—for example, by distributing the code that defines a component's behavior across many programming modules, and by modifying the behavior of the solution algorithm based on the status of one or more components. This makes it difficult to understand and update existing components, and difficult to add new components.

On the face of it, these amount to programming concerns, nothing more. That is, a programmer familiar enough with the code can implement any model that meets the intrinsic restrictions of the program. However, the difficulties associated with adding new models go beyond mere programming matters, because they lengthen the development cycle sufficiently that COMIS cannot serve as an effective tool for research on the models themselves.

For this discussion, divide airflow and pollutant transport researchers into two groups: those who develop component models, and those who simulate systems using previously-developed models. Computational experience at the Lawrence Berkeley National Laboratory shows that only the simulationists use COMIS. Model development takes place exclusively in other simulation environments—for instance, SPARK for zonal models, Matlab for dynamic duct models, and purpose-produced code for dynamic pollutant transport models.

This distinction between modelers and simulationists would not exist if COMIS provided a natural platform for expressing, modifying, and testing new models. If it did, the model researchers would adopt COMIS, since it would: (1) support their work by providing tested models of other system components; and (2) increase the likelihood that their models would pass directly into the hands of users.

5.4. Restricted flow form. Flow components must take the form $f_{1\text{-}2} = f_{1\text{-}2}\{p_1 - p_2\}$. That is, they must calculate a steady-state flow as a function of the pressure difference across a component's terminals. Beyond this, the flow must not decrease as the pressure drop across the element increases.

Flow expressions of this form yield symmetric system matrices. If, in addition, the flow between zones increases monotonically with the pressure drop, the system yields positive-definite matrices [14]. COMIS uses the symmetric positive-definite property to simplify its data structures, matrix factorization methods, and nonlinear solution algorithms considerably. Thus, it requires steady-state flow models that give flow as a nondecreasing function of the pressure drop.

* Calling an external CFD program from within COMIS, and incorporating its results into the remaining calculations, would remove this objection. However, this approach still would require modifying the program wherever it invoked the well-mixed zone model, and might introduce convergence problems in the airflow system.

Steady-state flow elements. Two common steady-state flow components, duct junctions and fans, violate these requirements.

Junctions, as three-port flow elements, break symmetry. While nominally COMIS provides a junction flow model, it cannot solve systems containing one. Of course, a modeler can implement a duct junction as a zone, and attempt to account for junction losses using the duct model. However, this approach misses the fact that the dissipated energy in each branch of a junction depends on the flows in the other branches [15 §11-8]. Changing the flow in one branch can even produce suction on a branch that previously faced a retarding pressure. In effect, this would change the sign of the loss coefficient in the corresponding duct. Therefore if flow rates can vary, and especially if air can reverse through any branch of the junction, then only a three-port junction model can account for the losses correctly.

The other problematic element, the fan, typically has a negative slope over some portion of its pressure-flow curve. Besides violating the requirement for nondecreasing flows, this creates a low-flow region where some pressure drops have two or three associated flows [15 §11-2]. COMIS overcomes this problem by replacing the low-flow part of the actual fan curve with an approximate, monotone-increasing curve that yields a unique flow for every pressure drop [7]. If a simulation involves only design flow rates, this numerical device does not affect the calculated results.

Zonal models. As noted above, to model details of airflow in a room without changing the internal structure of COMIS would require representing the room as a collection of well-mixed subzones, each characterized by a single state variable. Unlike computational fluid dynamics approaches, zonal models meet this requirement. However, the flow elements used by the zonal model to connect its subzones also would have to satisfy the restrictions imposed by COMIS.

Zonal models draw strongly on a standard COMIS flow model, the orifice [11]. Unfortunately, this flow element does not properly express the physics of airflow in rooms [1]. This probably explains the poor performance of zonal models in predicting room airflows [16].

Other published zonal model elements simply prescribe the flows, based on fixed conditions such as wall temperatures and jet mass flow rates [11]. Models based on prescribed flows present little computational difficulty. In particular, COMIS can enforce a known flow between two subzones, so long as each also has a variable-flow connection [14]. Naturally, models that rely on prescribed flows will have limited application in cases where the room interacts strongly with the rest of the building.

Of course, the zonal approach admits other flow relations than orifice models and prescribed flows. COMIS can support them so long as they express the flow as a function of the pressure difference between the subzones. However, an inherent numerical difficulty will arise in any such model. Intuition suggests that the pressure difference between two points in a room will be small compared to the corresponding range of possible flows between those points. Thus, limitations of finite-precision arithmetic may dominate the flow model's behavior. Specifically, the smallest machine-resolvable change in a subzone's reference pressure could create a large change in the calculated flow. In this case, COMIS may not be able to achieve mass balance.

5.5. No dynamic airflows. As stated above, COMIS imposes a steady-state airflow model. That is, the functional form $f_{1\text{-}2} = f_{1\text{-}2}\{p_1 - p_2\}$ does not admit dynamics in the airflow system, for example to account for the momentum of air in a duct.

This modeling capability might be important in a building that attempts to limit the sudden spread of pollutants through its ventilation system, for example by closing a damper. In such a case, the momentum of air already moving through the duct would create greater pressure drops across the damper, and hence greater flows, than a steady-state model would predict.

This duct-damper system has another dynamic of interest—the time needed to close the damper. For this, COMIS could use a quasi-steady damper model, which would take a steady-state model of a variable-position damper, and change its position setpoint over the time period of interest. This would move the damper, in simulated time, through a series of steady-state operating points. While implementing a quasi-steady damper model in COMIS would require some work, it would not affect the airflow calculations, and hence would not violate the expected flow model form.

Given such a quasi-steady damper model, some flow systems will not require a dynamic duct model. If the damper closes much more slowly than it takes for the duct to come to steady-state

after an instantaneous change in the damper position, then the damper model alone would account for the dynamics of interest.

Similar comments apply to duct-fan combinations. Suppose the start-up or spin-down time of the fan dominates the ventilation system response, and depends mainly on dynamics in the fan's rotor and motor. Then purely steady-state airflow elements may suffice. The quasi-steady simulation would use a steady-state model of a variable-speed fan, and change the fan speed from one time step to the next.

Dynamic sensor models fall into the same category as damper actuators, and other time-dependent elements that do not directly involve airflow calculations. Incorporating such elements in COMIS might require considerable programming effort, but would not violate its restrictions on the flow element models. In fact, CONTAM has a simple event-based controls model.

Transport time. The dynamics considered here involve only the program's airflow calculations. Another, possibly more important, dynamic effect involves the transport time in flow elements.

COMIS does not account for the time pollutants spend in flow paths. While it uses the calculated mass flows to determine the total amount of pollutant transported between zones at each time step, it assumes that transport occurs instantaneously. Thus, the program overestimates the speed at which a ventilation system spreads pollutants throughout a building.

A natural approach to modeling transport time in flow paths involves breaking up a flow element up into multiple pieces, connected in series. In a sense, COMIS already supports this approach. The modeler can replace a single flow element with n new elements, linking $n-1$ new zones whose volumes sum to the presumed volume of the flow path. This technique can greatly increase the number of flow elements and zones in a simulation. It also requires the modeler to adjust the flow element parameters, so that an arbitrary pressure drop of $\Delta p_{a\text{-}b}/n$ across a single element produces the same flow as a pressure drop of $\Delta p_{a\text{-}b}$ would have produced in the original flow path.

A more elegant solution would implement this approach directly in the code that calculates pollutant transport. This would allow the user to estimate transport times without introducing new zones and links to the airflow system. In addition to reducing the size of the airflow system, this would eliminate the need to re-parameterize the affected flow elements, and would ease the task of establishing the appropriate number of divisions for each flow path.

No effort has been made to investigate the difficulty of implementing any transport time model in COMIS. However, since it would affect only the code that calculates pollutant dispersion, it would not violate the requirement for steady-state airflow models.

5.6. Nonrepeatable calculations. Flow models in COMIS can give nonrepeatable results due to "memory" in the calculation routines. In order to reduce the computational burden incurred by repeated evaluations, some flow routines store intermediate results for use during the next evaluation of the flow path in question. For these models, the flow resulting from one evaluation of a particular element may depend on the value calculated at the last call of the defining routine.

Memory in a flow routine has two negative consequences. First, the routine can find flows incompatible with the defining equations. Second, repeated evaluations of the flow element with the same input pressures can result in different flows.

Inconsistent with definition. A flow element routine that depends on values from previous calculations will, at some point, find flows incompatible with its defining equations. For example, the routine implementing the crack model in COMIS finds the path density, ρ, using the flow calculated during the last evaluation of the crack in question*. It then uses this density to find the new flow. However, it does not require that the new flow and density jointly satisfy the density relation. Thus, the routine can return flows that violate the crack model's defining equations (even if no errors arise due to incorrect coupling of the mechanical energy balance).

Fortunately, in most cases the deviation from the model is likely to be small compared to uncertainty in the model. In other words, any errors induced by memory in the flow routines probably fall below the user's expectations of the accuracy with which the model predicts the

* In the duct model, this would be equivalent to finding ρ by using the last calculated velocity in Equation 4.4. The actual duct model sets ρ to either ρ_1 or ρ_2, depending on the sign of Δp_L.

flows in a real building. Furthermore, as the airflow solver converges, it tends to make only small adjustments to the estimated pressures. Hence, flow elements with memory often converge to values exactly consistent with their defining relationships.

Nonrepeatability. If a flow routine's calculations depend on previous results, then repeated evaluation with the same set of input pressures may not produce the same flows. In practice, the nonrepeatability of flow evaluations has greater consequences than the simple fact of inconsistency with the defining equations—it can affect the algebraic solver responsible for finding a solution to the steady-state airflow problem.

Iterative methods for solving nonlinear algebraic systems generate a sequence of trial solutions for the system. Many iterative methods do not accept a new trial step automatically. Rather, they compare the results of each trial step to those of the last accepted step, both in order to accept or reject the trial solution, and in order to control the selection of the next trial point [6 §6.5].

In COMIS, this means comparing the overall flow imbalance for different sets of trial pressures. When flow routines can deliver different flows at the same pressures, the solver cannot meaningfully distinguish which of two slightly different trial solutions produces smaller flow imbalances. In some cases, nonrepeatable flow evaluations can stagnate such a solver, as it repeatedly cuts the step length in an attempt to reduce the flow imbalances [13].

The effect of nonrepeatable flow evaluations on the solver can be interpreted in terms of the derivatives of the flow relations. The solver requires continuous flows, with continuous, bounded first derivatives with respect to the pressures [14]. When a flow can change merely as a result of re-evaluating at the same pressures, then the flow function is not continuous. Furthermore, the change in flow resulting from a slight perturbation to a zone pressure may bear no relation to the derivative of the flow model.

5.7. Programming language. COMIS is coded in Fortran-77, which lacks both dynamic memory management and support for object-oriented programming.

Memory allocation. COMIS, coded in Fortran-77, necessarily allocates all memory statically—that is, at the time the program starts up. This means the programmer, not the modeler, sets the size of every array. Hence it is possible to define a simulation with more zones or links than the program supports. In this case, it is necessary to increase the hard-coded number of zones or links available, and build a new executable.

Clearly this scheme "wastes" computer memory when running the program on problems much smaller than those for which the executable was built. Worse, if the environment used for building an executable limits the size of the program stack, then it places absolute limits on the size of problems the program can simulate. The Airflow and Pollutant Transport group at Lawrence Berkeley National Laboratory has encountered this limitation, for executables built using Compaq Visual Fortran v. 6.1 on Windows machines. This build environment limits executables to something in the neighborhood of 1000 zones and 5000 links (of course, the balance between zones and links affects the memory use).

Object style. In an object-oriented language, a programmer-defined *class* combines both variables (which contain data) and functions (which operate on data). Thus the programmer can bind together the data and functions that support a certain model [5 §9]. This supports strongly modular simulation by: (1) collecting all model-related code together; and (2) restricting a model's interactions, with both the simulation environment and other models, to a few well-defined exchanges of data.

Object-oriented languages also support *inheritance*, allowing the programmer to create parent classes from which all child classes acquire variables and functions [5 §14]. Thus for example 'duct' and 'crack' models could inherit behavior from a parent class of generic 'flow elements'. This would reduce the effort required to add new models, for example by providing routines to read the model parameters from an input file, link the model to the system, or estimate the model derivatives using finite differences. More important, it would force new models to conform to expected coding standards. Such standards might govern what routines a new model had to provide, the types of data those routines accepted and returned, or the assumptions it could make about the other models with which it interacts.

Another advantage of inheritance becomes apparent when adding new flow models to the code base. Fortran-77 requires a decision tree at every point where COMIS loops through all the flow elements in a system. The decision logic checks the type of the current flow element (e.g., duct, crack, etc.), and calls the appropriate routine. To add a new flow model, the programmer must add a new test to every such decision tree. These trees become quite long, and they occur in a number of places in the code. An object-oriented program would contain a single call to a generic flow element routine at every such point, and would select automatically, at run-time, the routine appropriate for the flow component currently at hand.

Fortran-90. Fortran-90 provides many features not found in Fortran-77, including dynamic memory allocation, user-defined data structures, and provision for a modular programming style. Furthermore, its compatibility with Fortran-77 means it would provide a relatively straightforward upgrade path for COMIS. However, it does not go far enough toward object orientation. In particular, it does not support inheritance.

5.8. Ownership. The most recent version of the program, COMIS v. 3.1, is administered by the Swiss Federal Laboratories for Materials Testing and Research (EMPA). The distribution fee (about $400) supports activities such as fixing errors, updating the user guide, improving the graphical interface, and providing user support.

From a user's point of view, this is a positive development. It does, however, complicate the question of ownership of the code. Researchers who plan to modify the source code extensively (i.e., to the point where they end up with a substantially different program) may prefer to work from an unlicensed, "free" code base—that is, from COMIS v. 3.0.2.

Obviously, researchers will prefer to use COMIS v. 3.1 whenever possible—not only because of its superior support, continuing development, and bug fixes, but also because staying in step with the rest of the COMIS community increases the ability to share models to others. However, when the possibility exists that the researcher may have to "fork" the program, creating a distinct version, then these advantages may carry less force. In that case, the desire for clear ownership of the forked code may make COMIS v. 3.0.2 more desireable as a starting-point for development.

Where researchers wish to combine COMIS with other programs, for example the aerosol transport and fate model MIAQ4, other licensing issues will come into play. For example, the GNU General Public License (GPL), under which MIAQ4 is released, requires that any program containing or derived from MIAQ4 must also be distributed under the terms of the GPL [9]. The legal implications of the GPL go beyond the scope of this document.

6. Recommendations

In general, three approaches exist for addressing deficiencies in an existing program: (1) make ad hoc changes, introducing code patches and decision trees to modify the program's behavior on a case-by-case basis; (2) restructure the code, to reimplement or regularize its behavior in a consistent manner for all cases; or (3) find or write a more capable program.

Given the known restrictions already embedded in the code for COMIS, the limited extensibility of its framework, the difficulty of modifying its significant monolithic parts, and the licensing questions, the Airflow and Pollutant Transport group at Lawrence Berkeley National Laboratory should seek a more flexible, more capable program for pursuing its research on airflow and contaminant transport.

Because COMIS can model a useful range of building airflow and pollutant transport problems, is reasonably efficient, and has a strong user base, the Laboratory should not abandon COMIS. However, in light of the constraints that COMIS imposes on airflow models, and its limited support for new models, the Airflow and Pollutant Transport group should take steps to identify, develop, and gain experience with a viable alternate simulation environment.

A number of obvious alternatives exist. Most notably, CONTAM, SPARK, and IDA were all developed specifically with buildings applications in mind. It would also be possible to create a completely new simulation framework. In addition, other general-purpose simulation environments must exist—in particular ones with no licensing issues, that support (or even enforce) strongly

modular simulation.

6.1. CONTAM. COMIS closely resembles the CONTAM multizone airflow simulation program, developed at the National Institute of Standards and Technology [19]. CONTAM generally is perceived as easier to upgrade than COMIS—largely because it is written in a more modern programming language (C as opposed to Fortran-77), and because of its more consistent code (written by one programmer rather than by many). Thus, the program inevitably comes up when discussion turns to a successor for COMIS.

Unfortunately, CONTAM shares many of the design features that make COMIS a questionable platform for long-term development. In particular, CONTAM also: (1) provides only well-mixed zones; (2) restricts the flow components to steady-state, monotone increasing functions of the pressure drop; and (3) decouples the mechanical energy balance from the flow equations, though in a slightly different way than does COMIS. In addition, some flow components in CONTAM also have memory, so that the results of one evaluation influence the flow calculated at the next evaluation.

6.2. SPARK. SPARK, the Simulation Problem Analysis Research Kernel, provides a general-purpose environment for simulating differential-algebraic systems [4], [18]. As with COMIS, the user defines a simulation problem by specifying the connections between components. However, in SPARK, the user also defines the component models.

Defining models. To create SPARK components, the modeler first writes the more fundamental *atomic* objects, which represent individual equations. Ideally, the modeler provides multiple implementations of each atomic equation object, giving various assignment expressions derived from the algebraic relation. For example, a mass continuity relation for zone 1, $f_{1\text{-}2} + f_{1\text{-}3} = 0$, could yield the assignments $f_{1\text{-}2} := -f_{1\text{-}3}$ and $f_{1\text{-}3} := -f_{1\text{-}2}$.

The modeler may bundle multiple equations together to form components. However, this is strictly for convenience—fundamentally, SPARK treats the entire system at the equation level.

SPARK processes a simulation model by constructing a data flow graph to show how the variables in the assembled equations depend on one another. It applies graph theory, choosing from among the assignment expressions in order to reduce the number of independent variables in the system. It can also decompose the problem into independent subsystems. For example, given equations that defined a COMIS-style system, SPARK could separate the equations defining pollutant transport from those defining the airflows, solving the transport problem only after calculating the airflows.

After determining the order in which to evaluate the equations, SPARK creates an executable program tailored to the simulation model at hand. The user runs this executable, possibly supplying run-time parameters to complete the model realization.

Implementation issues. SPARK aims to provide a completely general modeling environment, and one that eases the task of defining new simulation components. Such an environment would greatly aid researchers interested in quickly writing and testing new component models—for example to explore novel control strategies for a building, to add a previously unsupported feature to an existing model, or to scope experimental requirements when developing a new model.

The program remains, moreover, a work in progress. Its use of graph theory to sort the equations, plus the fact that this process generates the complete source code defining a simulation, means that SPARK requires a large amount of memory to process and compile a simulation into an executable. Perhaps more seriously, its numerical solution algorithms have shown convergence problems. Furthermore, processing the assignment expressions using graph theory can make the final selection of variables, and their dependencies, somewhat opaque to the user (though SPARK does provide tools for both determining and influencing the final selection of variables). This opacity can make troubleshooting component models, or simulation models, more difficult.

Finally, it is not clear that SPARK supports models with variable-length array inputs. This would make it problematic to define a general zone model, for example, since the number of flow elements connecting to each zone can vary.

Impending releases of SPARK should address at least some of these problems. For more information, see `http://simulationresearch.lbl.gov/VS/spark.html`.

6.3. IDA. The simulation environment IDA also provides a general framework for defining and solving differential-algebraic systems [17]. The literature describing IDA also lays out an impressive range of ideas regarding desireable qualities in a general simulation program.

IDA is a commercial program, of note here mainly because of: (1) its origins in, and existing library of models for, building energy simulation; and (2) its use of Neutral Model Format to express models equationally, rather than algorithmically [3].

For more information on IDA, see `http://www.equa.se/eng.ida.ice.html`.

6.4. Purpose-produced alternative. It would be possible to create a simulation *framework*, that is, a program that mediates between user-defined component models and public-domain general differential-algebraic solvers such as DASSL [2 §5]. The framework would consist entirely of an interface designed to support strongly modular simulation.

The interface would define a set of functions through which the component models would interact with the simulation framework, and with one another. For example, in the duct model these functions would include: (1) procedures for creating a particular instance of a duct—setting parameters, making requests to the simulation framework for internal variables (such as the friction factor), and informing the framework about any connection variables (such as p_1 or $f_{1\text{-}2}$); (2) procedures for initializing the model at the start of each time step; (3) a procedure for evaluating the model while solving the system; and (4) procedures for reporting variables of interest at the end of a time step (for instance, the power dissipated in the duct).

This design would represent a more algorithm-oriented approach than that of SPARK or IDA. Possibly this would relieve the need for the up-front preprocessing time, and the large computer memory, that SPARK requires. On the other hand, it would tend to create larger, sparser Jacobian matrices than those of SPARK, with the attendant slower factorization times [18]. There is, however, anecdotal evidence to suggest that sparser systems can require fewer iterations overall to solve than the corresponding compacted, dense systems [4 p.286].

To speed the development of new models, the framework should include at least two distinct pieces: (1) a simulation engine, which reads and performs a simulation according to a specification that is well-formed in some sense; and (2) an input engine, which accepts a human-readable simulation description, checks it for errors, and converts it to a well-formed simulation specification for the simulation engine to perform. Separating the user interface from the simulation tool would allow model developers to code and test new components without first providing a user interface for the model inputs. Furthermore, it would allow greater flexibility in the user interface, for instance permitting a text-based input format to coexist with a graphical interface, and permitting different graphical interfaces for different platforms.

Such a limited simulation framework, once defined, would pose only problems of efficient implementation. However, in some ways it represents a greater risk than SPARK. Even making use of public-domain software for parsing input files, solving the nonlinear systems, and factoring or indirectly solving sparse matrices, the development of such a framework would still require a large investment in programmer time. The same time might be better spent implementing more robust solvers in SPARK.

6.5. Recommendation. An effort should be made to identify and compare simulation environments that can duplicate the capabilities of COMIS without imposing the modeling restrictions outlined above. The alternatives mentioned above were included, not in the spirit of advocating the adoption of one or the other, but to point out that a number of possibilities exist for replacing COMIS for multizone airflow and pollutant transport modeling.

In particular, efforts should be made to identify simulation environments that:

- support a strongly modular approach to defining component models;
- allow systems of mixed differential and algebraic equations;
- provide event-based models, for example to simulate a thermostat, or a ventilation system controller that switches modes based on the level of pollutant in a room;
- support direct and indirect solution methods suitable for coarse-grid CFD models;
- allow variable-length array inputs;
- provide an intuitive user interface for connecting component models in order to build up systems;

- encourage experimentation with component models and system-level combinations of component models, by providing fast turnaround when implementing new models; and
- support the transparent communication of models and solution techniques between researchers, for example by using public-domain source code and text-based mechanisms for defining models.

These environments should be evaluated in terms of their suitability for replacing COMIS. As a first step, the current capabilities of COMIS should be programmed as component models, and tested in simulations of building airflow systems. Note that in such a trial, one should not expect a general simulation tool to calculate exactly the same values as a comparable COMIS building model—mainly because it will not reproduce the same implementation details, such as the decoupled mechanical energy balance, imposed by the monolithic parts of COMIS.

Conversely, one should not expect a general simulation environment to match the execution speed of COMIS, because: (1) a general environment cannot employ as efficient a solution algorithm, since it does not impose the same restrictions on flow models; and (2) a general environment must couple all the equations properly, and hence must solve a larger and more complex system of nonlinear equations.

Acknowledgements. This work was supported by the Office of Nonproliferation Research and Engineering, Chemical and Biological National Security Program, of the National Nuclear Security Administration under U.S. Department of Energy Contract No. DE-AC03-76SF00098.

7. References

[1] James W. Axley, *Zonal models using loop equations and surface drag cell-to-cell flow relations*, in Roomvent 2000, Proceedings of the 7$^{\text{th}}$ International Conference on Air Distribution in Rooms, Reading, U.K. (July 2000), v. 1, pp. 235–240. Elsevier Science Ltd.

[2] K.E. Brenan, S.L. Campbell, and L.R. Petzold, "Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations." Society for Industrial and Applied Mathematics (1996).

[3] Axel Bring and Per Sahlin, *Modelling Air Flows and Buildings with NMF and IDA*, in Building Simulation '93, Proceedings of the Third International IBPSA Conference, Adelaide (August 1993), pp. 463–469. International Building Performance Simulation Association.

[4] W.F. Buhl, A.E. Erdem, and F.C. Winkelmann, *Recent improvements in SPARK: strong component decomposition, multivalued objects, and graphical interface*, in Building Simulation '93, Proceedings of the Third International IBPSA Conference, Adelaide (August 1993), pp. 283–289. International Building Performance Simulation Association. Lawrence Berkeley National Laboratory report LBL-33906.

[5] Guido Buzzi-Ferraris, "Scientific C++: Building numerical libraries the object-oriented way." Addison-Wesley Publishing Company (1994).

[6] J.E. Dennis, Jr. and Robert B. Schnabel, "Numerical Methods for Unconstrained Optimization and Nonlinear Equations." Society for Industrial and Applied Mathematics (1996).

[7] Helmet E. Feustel, *COMIS—an international multizone air-flow and contaminant transport model*, Energy and Buildings, v. 30 n. 1 (1999), pp. 3–18. Elsevier Science S.A.. Lawrence Berkeley National Laboratory report LBNL-42182.

[8] Andrew Ford, "Modeling the Environment: An introduction to system dynamics modeling of environmental systems." Island Press (1999).

[9] Free Software Foundation, `http://www.fsf.org/licenses/licenses.html`.

[10] Philip M. Gerhart and Richard J. Gross, "Fundamentals of Fluid Mechanics." Addison-Wesley Publishing Company (1985).

[11] Christian Inard, Hassan Bouia, and Pascal Dalicieux, *Prediction of air temperature distribution in buildings with a zonal model*, Energy and Buildings, v. 24 n. 2 (July 1996), pp. 125–132. Elsevier Science S.A..

[12] Y. Jaluria, S.H.-K. Lee, G.P. Mercier, and Q. Tan, *Transport processes across a horizontal vent due to density and pressure differences*, Experimental Thermal and Fluid Science, v. 16 n. 3 (March 1998), pp. 260–273. Elsevier Science Inc., New York.

[13] D.M. Lorenzetti and M.D. Sohn, *Improving speed and robustness of the COMIS solver*, in Roomvent 2000, Proceedings of the 7th International Conference on Air Distribution in Rooms, Reading, U.K. (July 2000), v. 1, pp. 241–246. Elsevier Science Ltd. Lawrence Berkeley National Laboratory report LBNL-44792.

[14] David M. Lorenzetti, "Computational Aspects of Nodal Multizone Airflow Systems." Lawrence Berkeley National Laboratory report LBNL-46949 (2001). Accepted for publication in Building and Environment.

[15] Faye C. McQuiston and Jerald D. Parker, "Heating, Ventilating, and Air Conditioning: Analysis and design, Third edition." John Wiley & Sons (1988).

[16] Laurent Mora, Ashok Gadgil, and Etienne Wurtz, "Comparing zonal and CFD model predictions of air flows in large indoor spaces to experimental data." Lawrence Berkeley National Laboratory report LBNL-47027 (2000). Accepted for publication in Indoor Air.

[17] Per Sahlin and Axel Bring, *IDA Solver: A tool for building and energy systems simulation*, in Building Simulation '91 Conference Proceedings, Nice, France (August 1991), pp. 339–348. International Building Performance Simulation Association.

[18] Edward F. Sowell and Philip Haves, *Efficient solution strategies for building energy system simulation*, Lawrence Berkeley National Laboratory report LBL-45936 (March 2000). Submitted to Energy and Buildings.

[19] George N. Walton, "CONTAM96 User Manual." National Institute of Standards and Technology, report NISTIR–6056 (September 1997).

A. Effect of decoupling the flow equations

As noted above, COMIS decouples Equation 4.1 from the flow element equations by setting the path density to some average of the zone densities ρ_1 and ρ_2. To estimate the flow error for a particularly sensitive case, consider a passive stack of diameter $D = 10\,\text{cm}$ and height $H = 6\,\text{m}$. Let the stack connect a room (zone 1), at 20 C, to the outside (zone 2), at 0 C.

The analysis makes the following assumptions:

(1) A properly-coupled set of equations would show that the stack fills with air from the room, so that $\rho = \rho_1 = 1.204\,\text{kg/m}^3$.

(2) COMIS uses a path density $\rho = \bar{\rho} = \frac{(\rho_1+\rho_2)}{2} = 1.248\,\text{kg/m}^3$ to find Δp_L, but uses ρ_1 to find the flow in the duct equations.

(3) The pressure drop through the stack from inside to out equals the pressure drop Δp_w through the building envelope from inside to out, plus the pressure drop due to hydrostatic effects in the ambient air,

$$p_1 - p_2 = \Delta p_w + \rho_2 g H \ . \tag{A.1}$$

(4) Parameters $\mu = 184.6{\cdot}10^{-7}\,\text{N}{\cdot}\text{s/m}^2$ and $\epsilon = 0.0015\,\text{mm}$ (a value for very smooth pipes [10 §7.2]).

It is not certain that COMIS will actually use $\bar{\rho}$ in the mechanical energy balance, since the logic controlling the density calculation does not admit straightforward analysis. Furthermore, the exact pressure drop in Equation A.1 depends on other particulars of the simulation. When the stack draws strongly enough, air will infiltrate through the walls, making $\Delta p_w < 0$ and decreasing the driving pressure $p_1 - p_2$. On the other hand, wind at the roof will create suction, increasing the pressure drop (but also making Δp_w more negative).

For the reference case, let $\Delta p_w = 0$. Following assumption 1, a proper coupling of the mechanical energy balance to the duct equations would give $\Delta p_L = 5.190\,\text{Pa}$ and $f_{1\text{-}2} = 0.02145\,\text{kg/s}$ (or a volume flow of about $0.018\,\text{m}^3/\text{s}$ or $1100\,\text{L/min}$). From assumption 2, on the other hand, COMIS calculates $\Delta p_L = 2.595\,\text{Pa}$ and $f_{1\text{-}2} = 0.01441\,\text{kg/s}$.

In this reference case, the flow error made by decoupling Equation 4.1 comes to about 33%.

The table below shows, for this reference case and several variations, the flow calculated under assumption 1, and the corresponding error made by COMIS under assumption 2.

Case	Flow, kg/s	Error, %
– Reference –	0.02145	32.8
$D : 10\,\text{cm} \rightarrow 30\,\text{cm}$	0.4142	32.1
$T_{\text{out}} : 0\,\text{C} \rightarrow -10\,\text{C}$	0.02762	32.7
$\Delta p_w : 0\,\text{Pa} \rightarrow -0.1\,\text{Pa}$	0.02121	33.6
$H : 6\,\text{m} \rightarrow 12\,\text{m}$	0.02145	32.8

In general, a variation on the reference case that increases the flow also decreases the relative error associated with using the wrong density in the mechanical energy balance. Therefore increasing the stack diameter reduces the relative error. However, the flow varies about its reference value more than does the error, and increasing the stack diameter increases the flow substantially. Thus this coding problem can make COMIS calculate large flows with large absolute errors.

The last row of the table shows that a taller stack yields the same flow as in the reference case. That is, under the listed assumptions, increasing the height of the stack has no effect on the flows. This occurs ecause the increased frictional losses in the duct exactly offset the greater driving pressure difference due to the hydrostatic effect. In an actual simulation, the pressure drop across the stack would depend on the calculated flows through all the interconnected zones, and changing the height of the stack would affect the flow through it.